Embedded Linux Conference Europe

29 October 2019

Writing Your Own Gadget with Zephyr OS

Andrei Emeltchenko

Contents

Introduction

Connection to PC & OS Drivers

Zephyr OS

Zephyr USB programming

Standard USB Classes

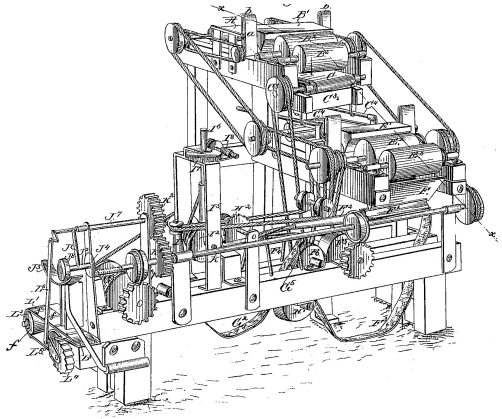
Ethernet over USB

Other USB Classes

802.15.4 over USB

WebUSB & USB/IP

USB

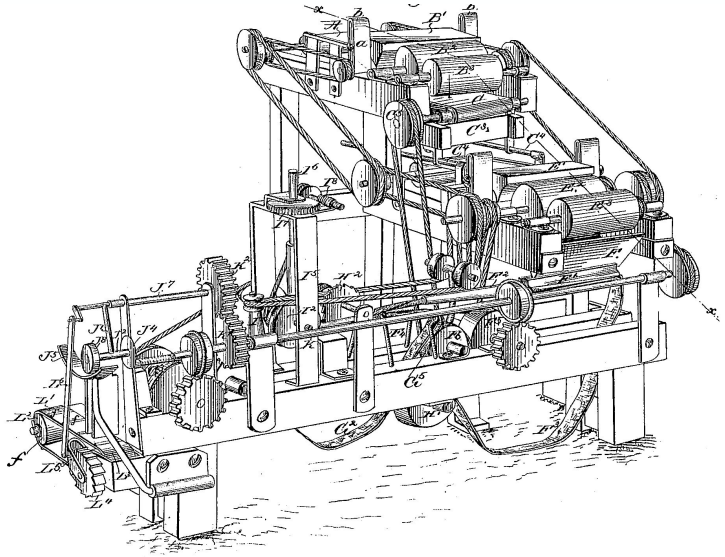


Complex Hardware

Introduction: Problem

- Many custom devices around (sensors, switches, etc)
- Problem connecting devices with non-standard interfaces to PC
  - SPI, I2C, etc

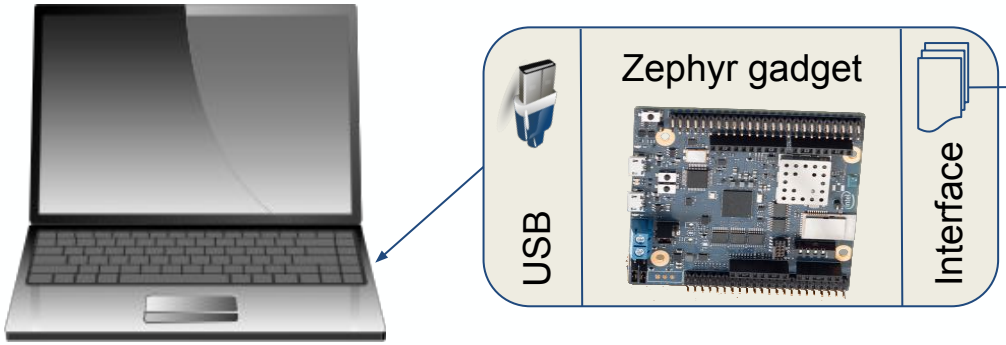
Access

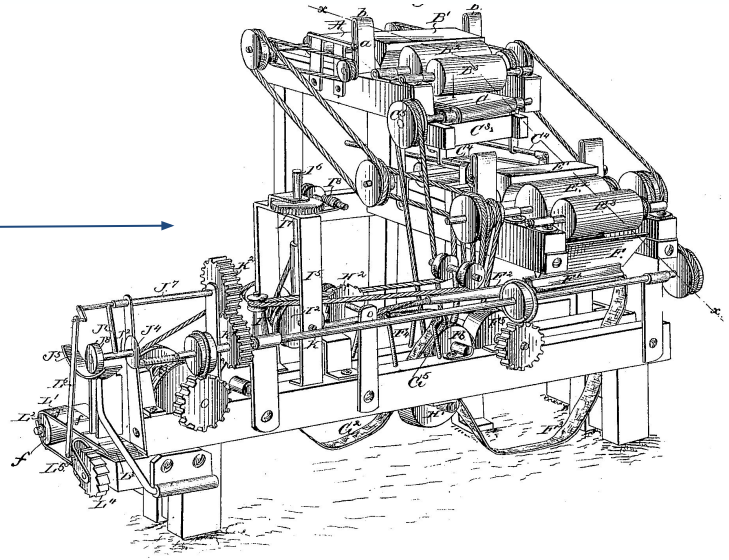


Complex Hardware

Introduction: Solution

- Use embedded board with right interface powered by Zephyr OS
- Zephyr board connects to Host via USB





Complex Hardware

Introduction: Zephyr

- Open Source OS for connected resource constrained devices https://www.zephyrproject.org/what-is-zephyr/
- Hosted by Linux Foundation
- Supported more than 100 boards: https://docs.zephyrproject.org/latest/boards/boards.html
- License: Apache 2.0
- Zephyr Project Documentation https://docs.zephyrproject.org/latest/index.html
- Source: https://github.com/zephyrproject-rtos/zephyr

Zephyr™ Project

Hello world in Zephyr

- Set up a development system https://docs.zephyrproject.org/latest/getting_started/index.html
- Set up build environment

```
$ source zephyr-env.sh
```

- Build hello_world sample for Qemu

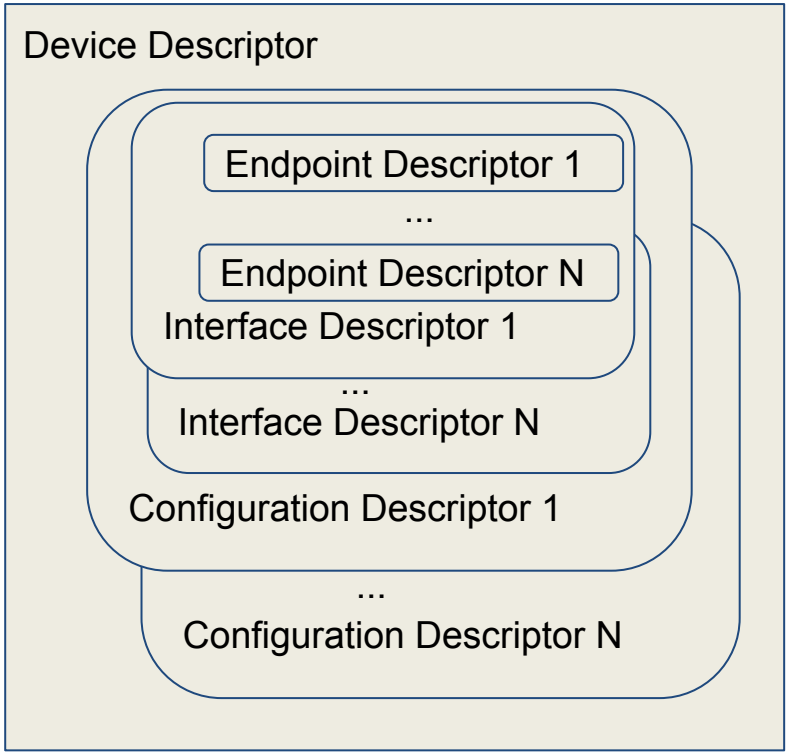
```
$ cd zephyrproject
$ west build –b qemu_x86 zephyr/samples/hello_world
```

- Run hello_world in Qemu

```
$ west build -t run

To exit from QEMU enter: 'CTRL+a, x'
[QEMU] CPU: qemu32,+nx,+pae

***** Booting Zephyr OS zephyr-v1.13.0 *****
Hello World! qemu_x86
```

USB: General overview

- One host connected to many devices
- Device identifies itself through descriptors
- Descriptors are binary data describing USB capabilities
  - Device Class
  - Product ID / Vendor ID
  - Configuration, Interfaces, Endpoints
- Endpoints are communication channels between host and device
- Standard Classes specified by USB IF
- Zephyr implements USB Device mode only

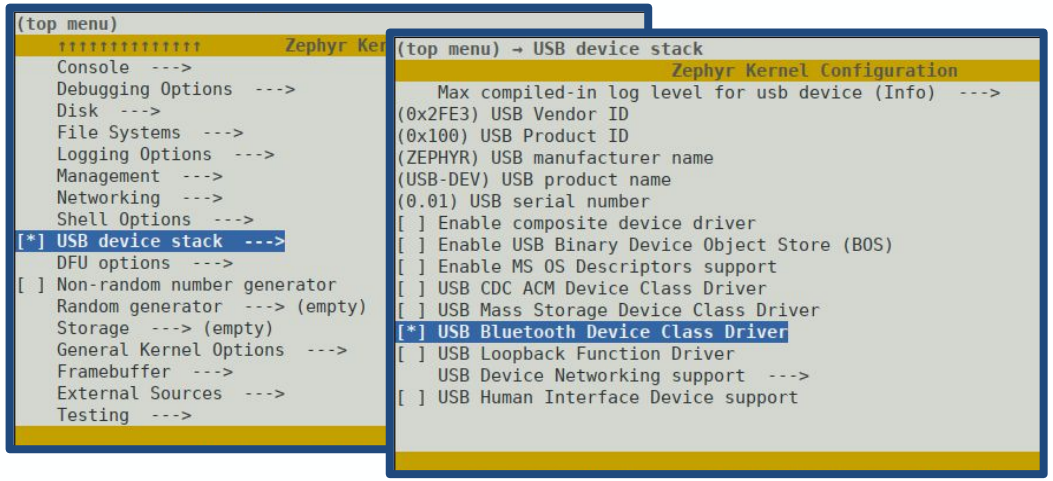


Programming USB gadgets: Standard Classes

- Basic USB Standard Classes are supported in Zephyr
  - Basic functionality is implemented in the Zephyr USB Device Stack
  - Some classes provide API (HID)
  - Others just work when enabled in application configuration
    - Bluetooth over USB sample (hci_usb)

```c
/*
 * Copyright (c) 2018 Intel Corporation
 *
 * SPDX-License-Identifier: Apache-2.0
 */

#include <zephyr.h>
#include <sys/printk.h>

void main(void)
{
    printk("Bluetooth over USB sample\n");
}
```

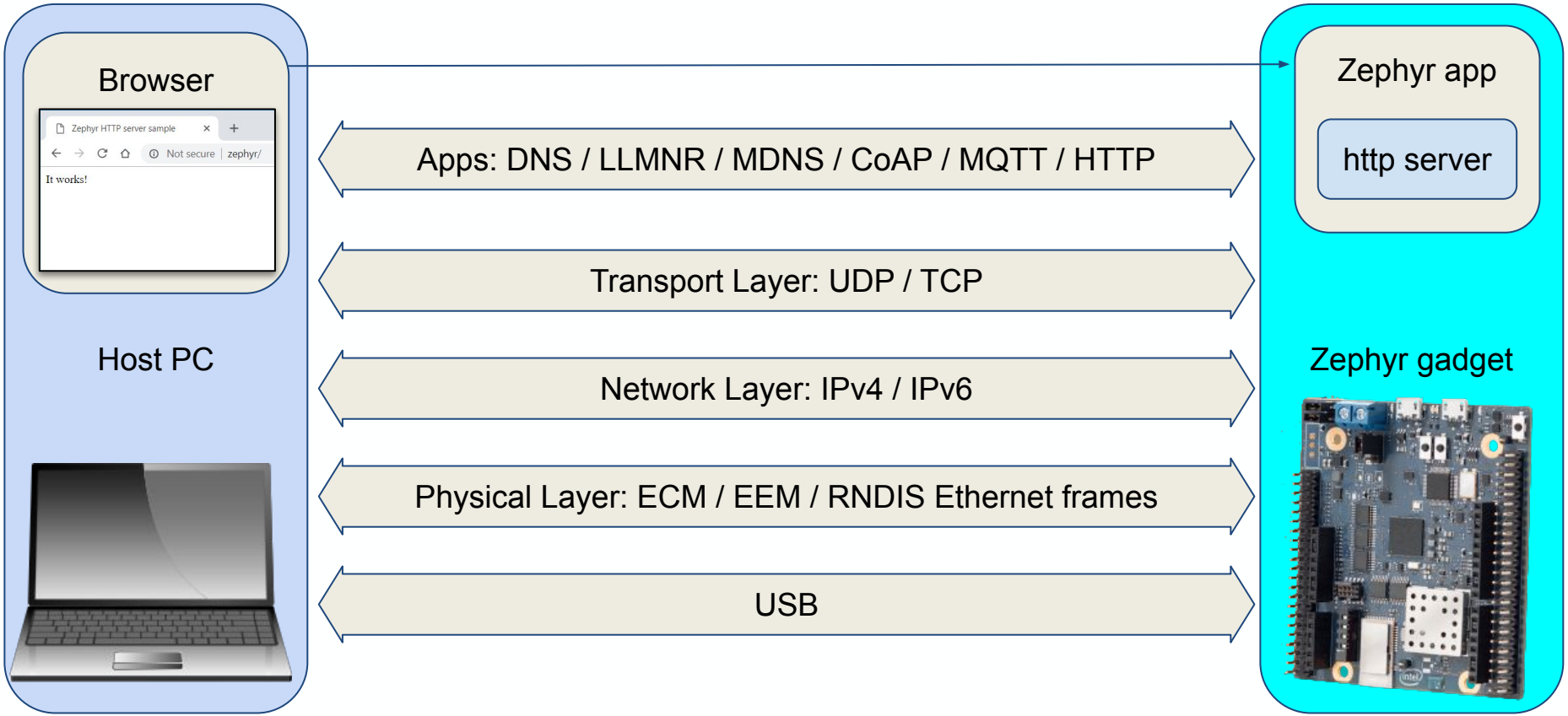
Source of samples/bluetooth/hci_usb



Supported Standard USB Classes in Zephyr

- Human Interface Device (HID)
- Device Firmware Upgrade (DFU)
- Mass storage over USB (MSC)
- Serial Port over USB (CDC ACM)
  - 802.15.4 “serial radio” protocol (wpan_serial)
    - 802.15.4 frames over Serial USB
    - Works with Contiki native border router application
- Bluetooth over USB
  - Bluetooth HCI RAW access is used to send HCI over USB
  - Bluetooth USB Transport Layer Spec. (Bluetooth Specs)
- Ethernet over USB
  - Standard protocols supported

Ethernet over USB: Use case

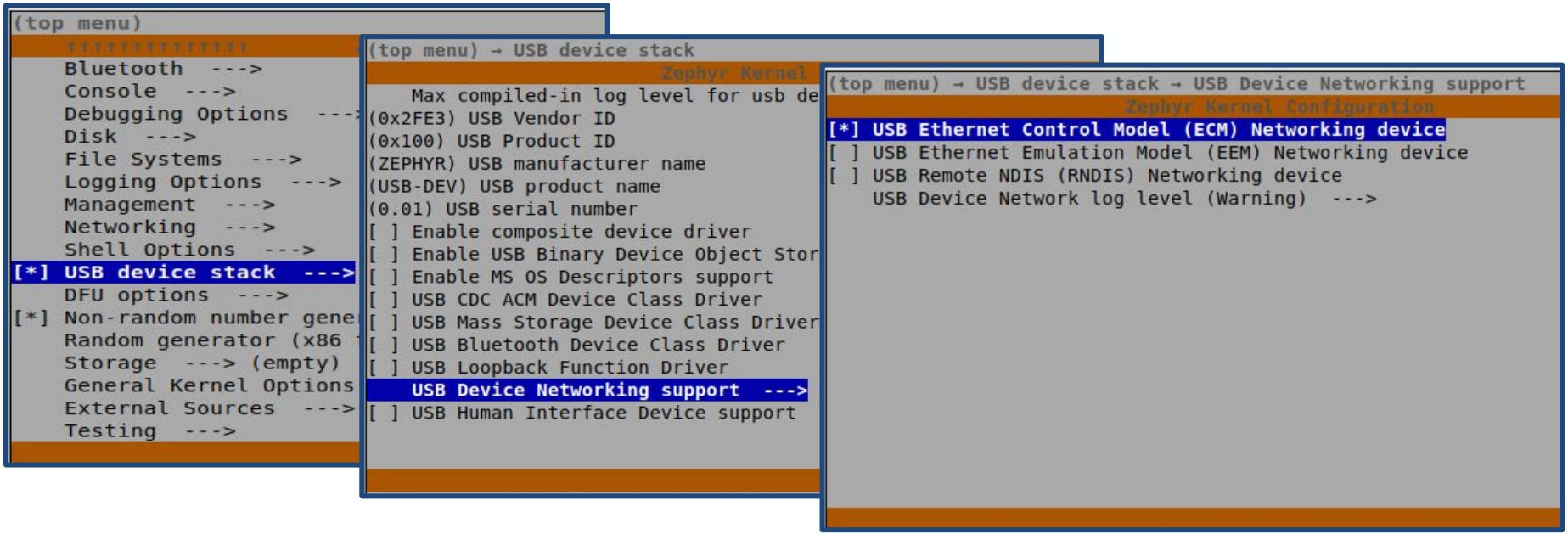


Ethernet over USB: Standards

- Motivation: Application independent data exchange
- Standards:
  - Microsoft Remote NDIS (RNDIS)
  - Communications Device Class (CDC) protocols
    - Ethernet Control Model (ECM)
    - Ethernet Emulation Model (EEM)
    - Network Control Model (NCM)

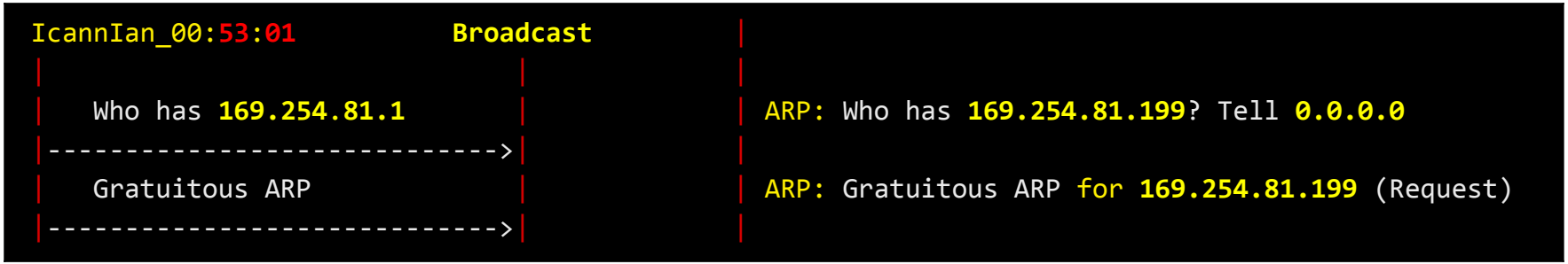
Ethernet over USB: Enabling in Zephyr

- Zephyr supports RNDIS, ECM and EEM
- Simple configuration: protocols are implemented in USB Device Stack
  - User only need to select Ethernet over USB checkboxes

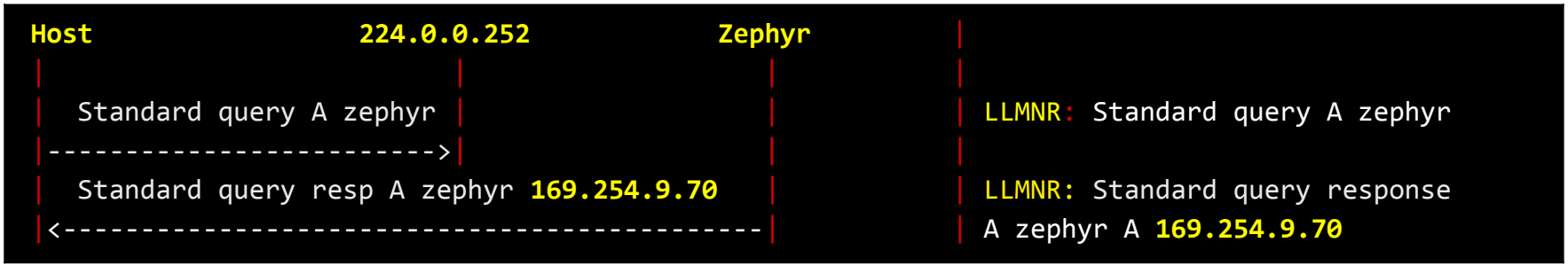


Ethernet over USB: Zero configuration

IPv4 address autoconfiguration for Host and Zephyr



Link Local Multicast Name Resolution (LLMNR)



zephyr can now be used as a hostname (Windows out of the box & Linux via systemd-resolved.service)

Ethernet over USB: HTTP server app

Zephyr **dumb_http_server** socket sample: source, README

Zephyr **dumb_http_server** + extra config

```
# USB Device settings
CONFIG_USB=y
CONFIG_USB_DEVICE_STACK=y

# Select USB Configurations
CONFIG_USB_DEVICE_NETWORK_ECM=y
```

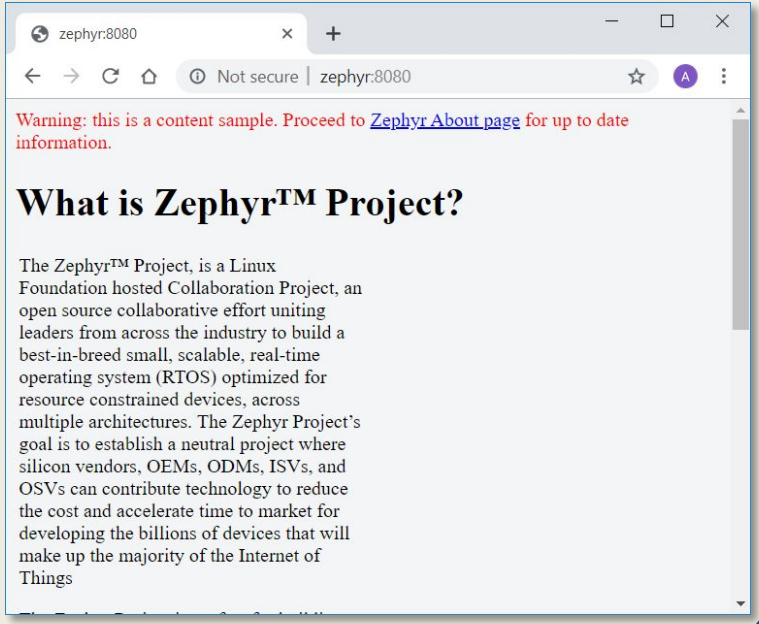
Ethernet over USB overlay config

```
# Zero Configuration
CONFIG_NET_HOSTNAME_ENABLE=y
CONFIG_NET_IPV4_AUTO=y
CONFIG_NET_CONFIG_SETTINGS=n
CONFIG_LLMNR_RESPONDER=y
```

Zero Configuration overlay config

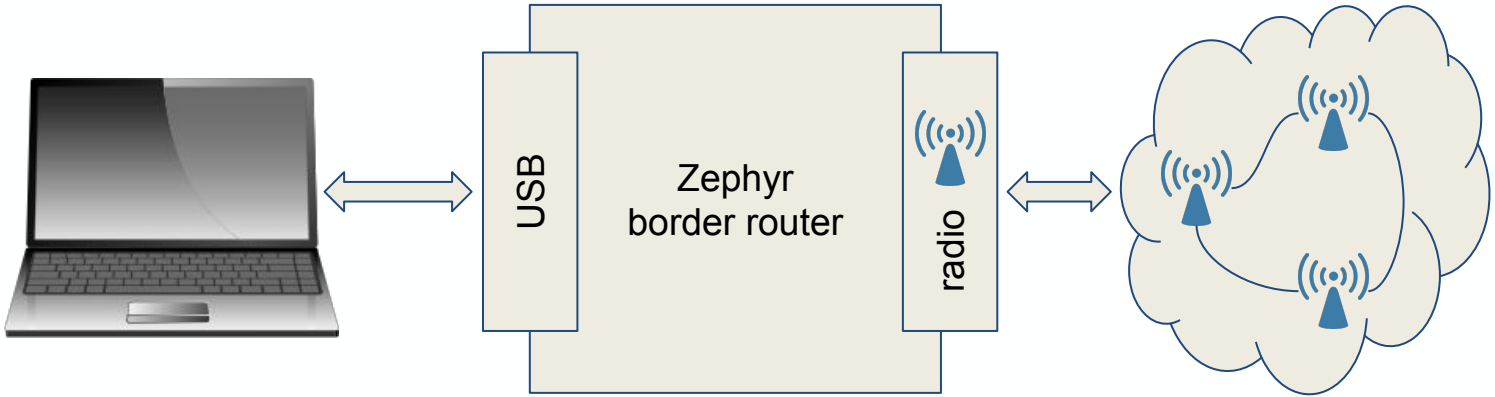
```
$ west build -b <board> <http sample> -DOVERLAY_CONFIG=
“overlay-netusb.conf;overlay-zeroconf.conf”
```

Host PC



Ethernet over USB: border router app

- Border Router connects networks with different routing domains
- Zephyr net capabilities: https://docs.zephyrproject.org/latest/guides/networking/
  - Multiple interfaces
  - Routing



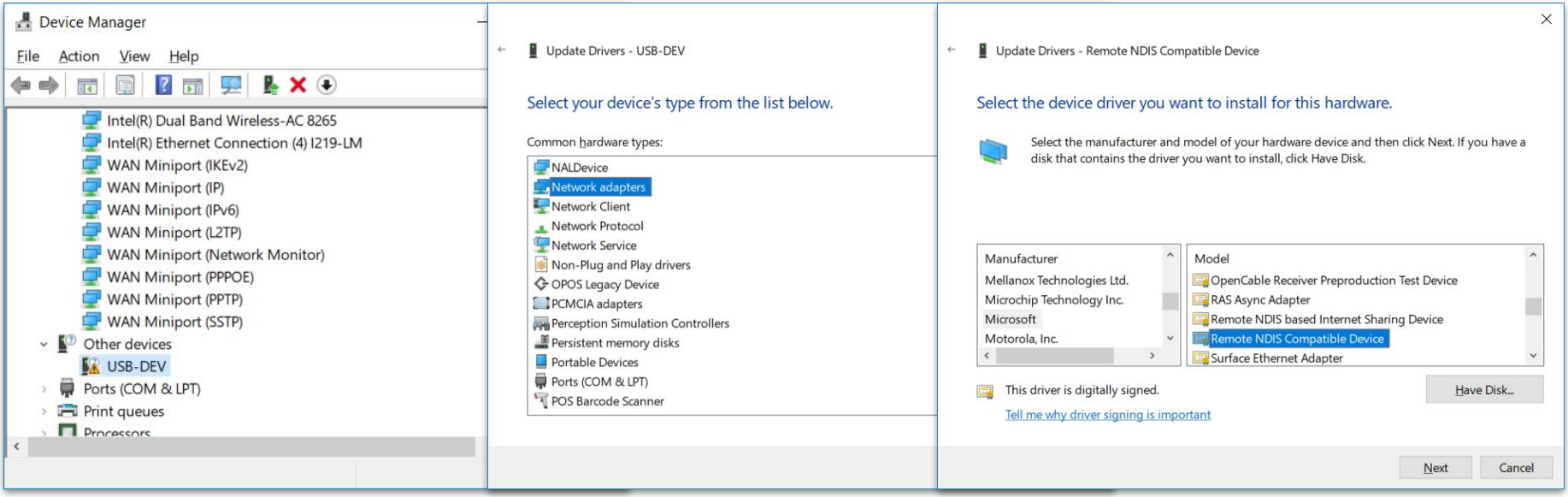
Ethernet over USB: Host OS Drivers

Vanilla OS	ECM	EEM	RNDIS
Linux	✓	✓	✓
macOS	✓	✗	✗
Windows	✗	✗	✓✗

Windows RNDIS support requires extra work

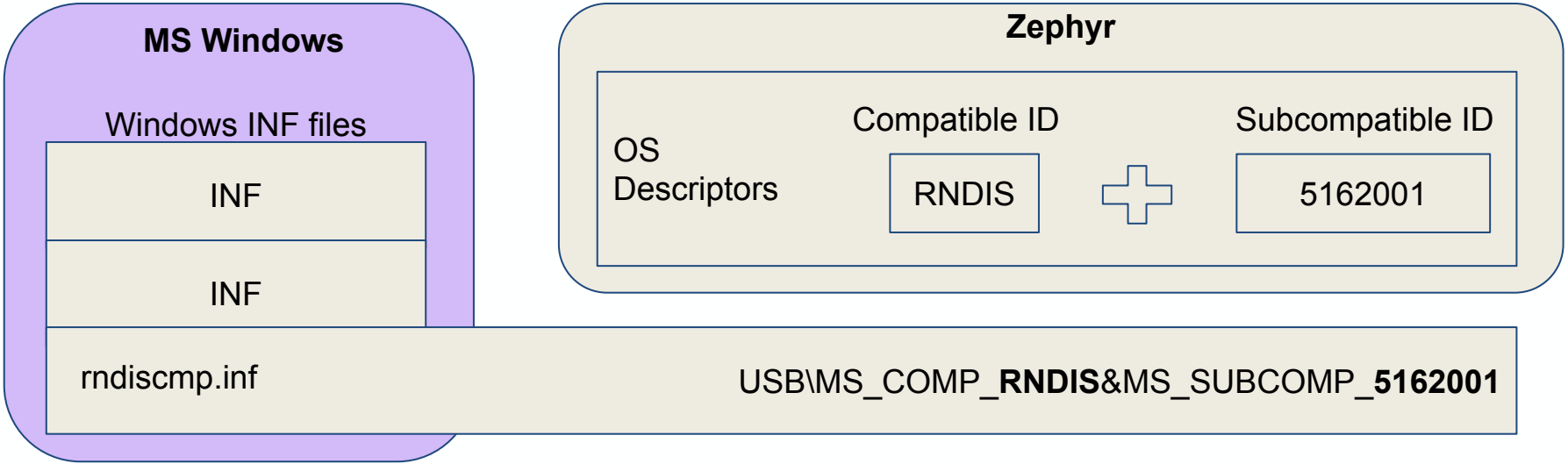
Ethernet over USB: Windows drivers

- RNDIS works in Windows after Device Manager manipulations
- The gadget may be recognized as Serial / COM depending on Win version
  - Device Class / Subclass matches usbser.inf (serial port)



OS Drivers for Zephyr USB: OS Descriptors

- Microsoft OS Descriptors help to autoconfigure MS Windows driver
  - Implemented for RNDIS at the moment
  - Tested with MS Windows 8.1 / 10



Compatibe ID: https://docs.microsoft.com/en-us/windows-hardware/drivers/install/compatible-ids

Programming USB gadgets: Custom Classes

- Custom class is not covered by USB IF specification
- The basics common to all classes are handled by USB Device Stack
- Application only need to define specific USB interface
- USB loopback implements custom interface
  - USB loopback interface defined: loopback source
  - Echoing data from one endpoint to another (callbacks below)
  - Used in testusb for USB testing: documentation
  - Linux driver available

```
static void loopback_out_cb(u8_t ep,
                            enum usb_dc_ep_cb_status_code ep_status)
{
    u32_t bytes_to_read;

    usb_read(ep, NULL, 0, &bytes_to_read);
    usb_read(ep, loopback_buf, bytes_to_read, NULL);
}
```

```
static void loopback_in_cb(u8_t ep,
                           enum usb_dc_ep_cb_status_code ep_status)
{
    if (usb_write(ep, loopback_buf, CONFIG_LOOPBACK_BULK_EP_MPS,
        NULL)) {
            LOG_DBG("ep 0x%x", ep);
    }
}
```

Programming USB gadgets: Custom Classes

Define interface and endpoints descriptors

```
struct usb_loopback_config {
    struct usb_if_descriptor if0;
    struct usb_ep_descriptor if0_out_ep;
    struct usb_ep_descriptor if0_in_ep;
} __packed;
```

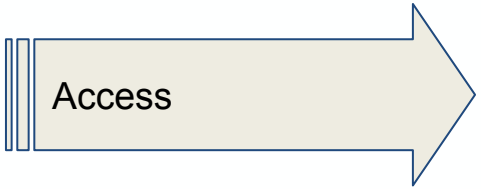
```
USBD_CLASS_DESCR_DEFINE(primary, 0)
struct usb_loopback_config loopback_cfg = {
    /* Interface descriptor 0 */
    .if0 = {
        .bNumEndpoints = 2,
        .bInterfaceClass = CUSTOM_CLASS,
    },
    /* Data Endpoint OUT */
    .if0_out_ep = {
        .bEndpointAddress = LOOPBACK_OUT_EP_ADDR,
    },
    /* Data Endpoint IN */
    .if0_in_ep = {
        .bEndpointAddress = LOOPBACK_IN_EP_ADDR,
    },
};
```

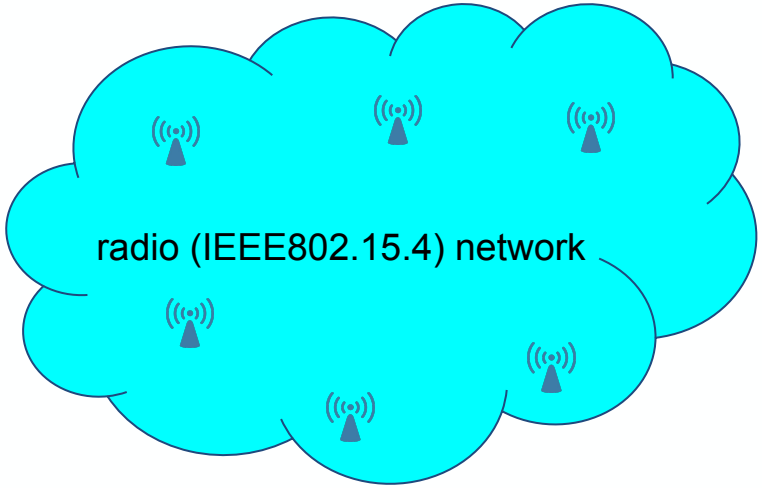
Define endpoints and configuration data

```
static struct usb_ep_cfg_data ep_cfg[] = {
    {
        .ep_cb = loopback_out_cb,
        .ep_addr = LOOPBACK_OUT_EP_ADDR,
    },
    {
        .ep_cb = loopback_in_cb,
        .ep_addr = LOOPBACK_IN_EP_ADDR,
    },
};
```

```
USBD_CFG_DATA_DEFINE(primary, loopback)
struct usb_cfg_data loopback_config = {
    .interface_config = loopback_interface_config,
    .interface_descriptor = &loopback_cfg.if0,
    .cb_usb_status = loopback_status_cb,
    .interface = {
        .vendor_handler = loopback_vendor_handler,
    },
    .num_endpoints = ARRAY_SIZE(ep_cfg),
    .endpoint = ep_cfg,
};
```

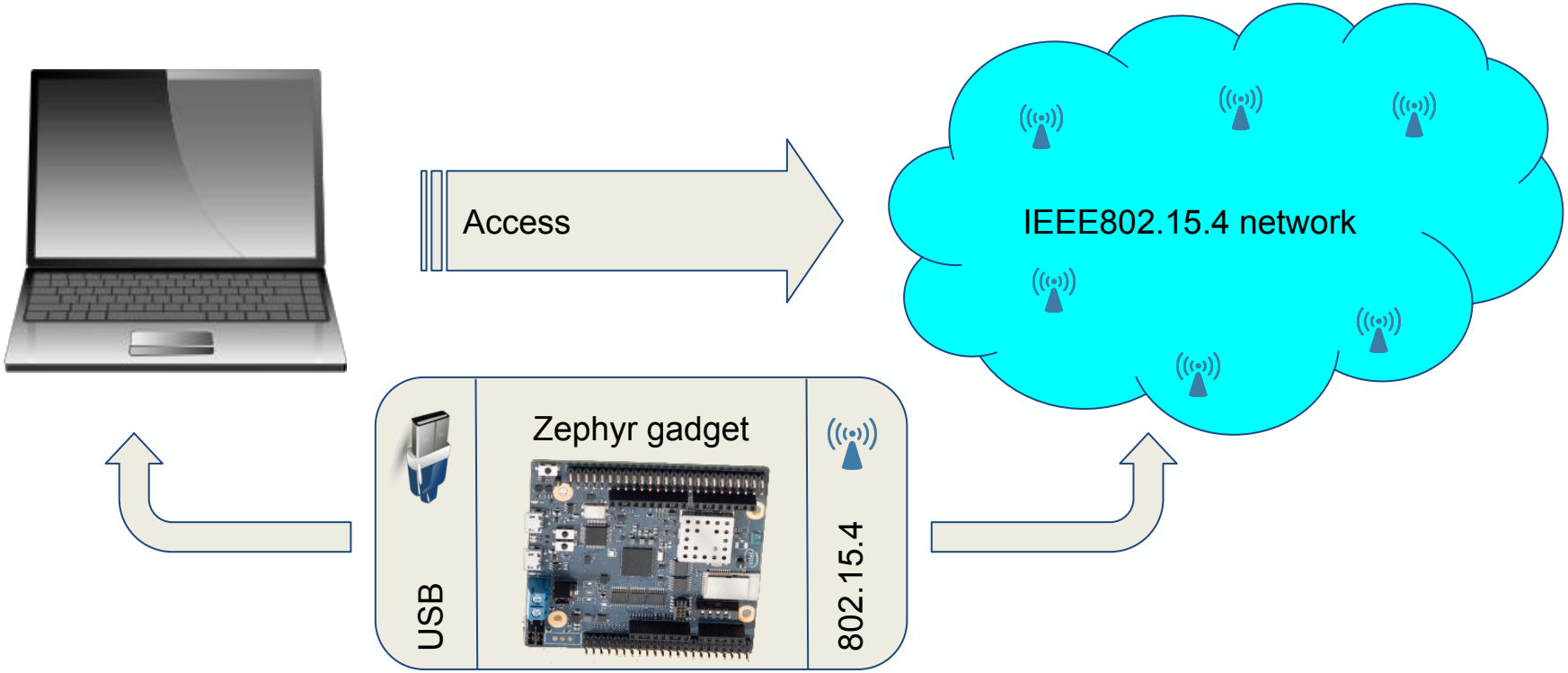
IEEE802.15.4 USB: Use case



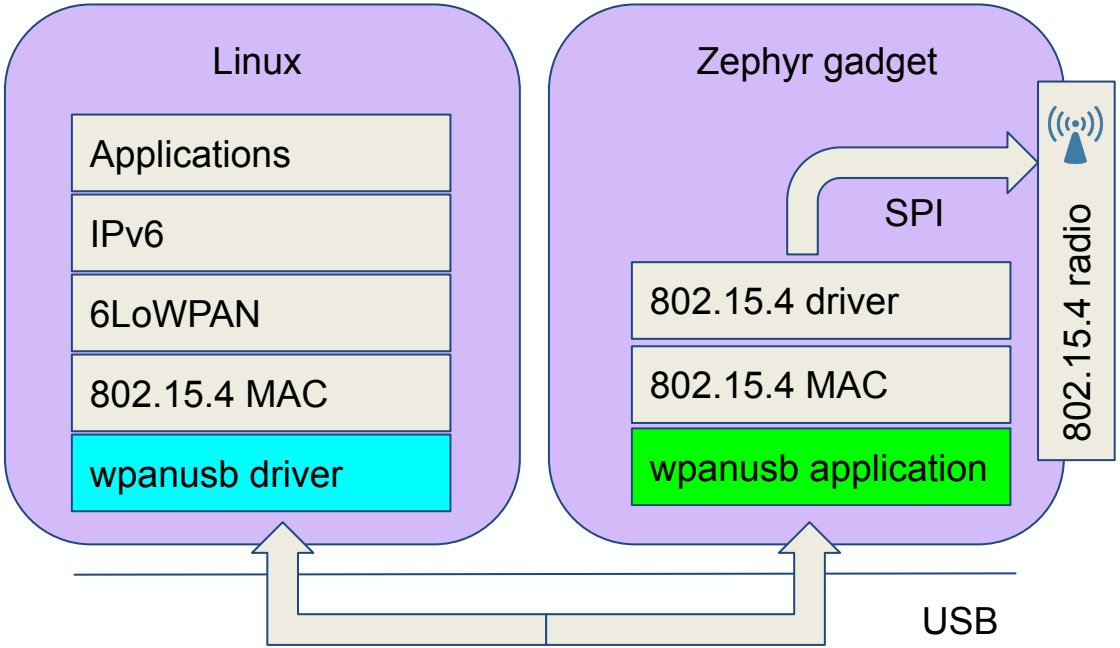


- Only one USB 802.15.4 adapter is supported by Linux (atusb)
- 802.15.4 SPI chips available

IEEE802.15.4 USB: Use case



IEEE802.15.4 USB gadget: Idea



IEEE802.15.4 USB gadget: API

Linux 802154 SoftMAC API

```
struct ieee802154_ops {
    ...
    int (*start)(struct ieee802154_hw *hw);
    void (*stop)(struct ieee802154_hw *hw);
    int (*xmit_async)(struct ieee802154_hw *hw,
                      struct sk_buff *skb);
    int (*set_channel)(struct ieee802154_hw *hw,
                       u8 page, u8 channel);
    int (*set_txpower)(struct ieee802154_hw *hw,
                       s32 mbm);
    ...
};
```

Zephyr 802154 Driver API

```
struct ieee802154_radio_api {
    ...
    int (*start)(struct device *dev);
    int (*stop)(struct device *dev);
    int (*tx)(struct device *dev,
              struct net_pkt *pkt,
              struct net_buf *frag);
    int (*set_channel)(struct device *dev,
                       u16_t channel);
    int (*set_txpower)(struct device *dev, s16_t dbm);
    ...
} __packed;
```

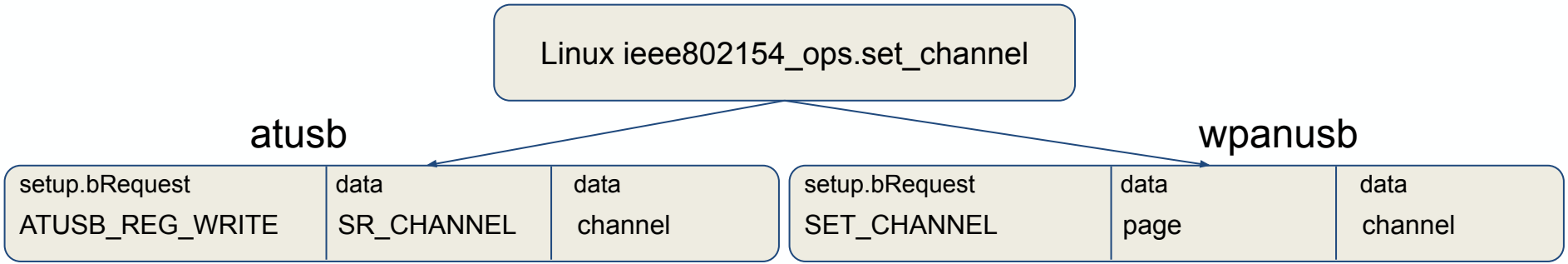
opcode: **SET_CHANNEL**
parameters: channel, page

setup.bRequest	data
opcode	parameters

USB

IEEE802.15.4 USB gadget: Zephyr application

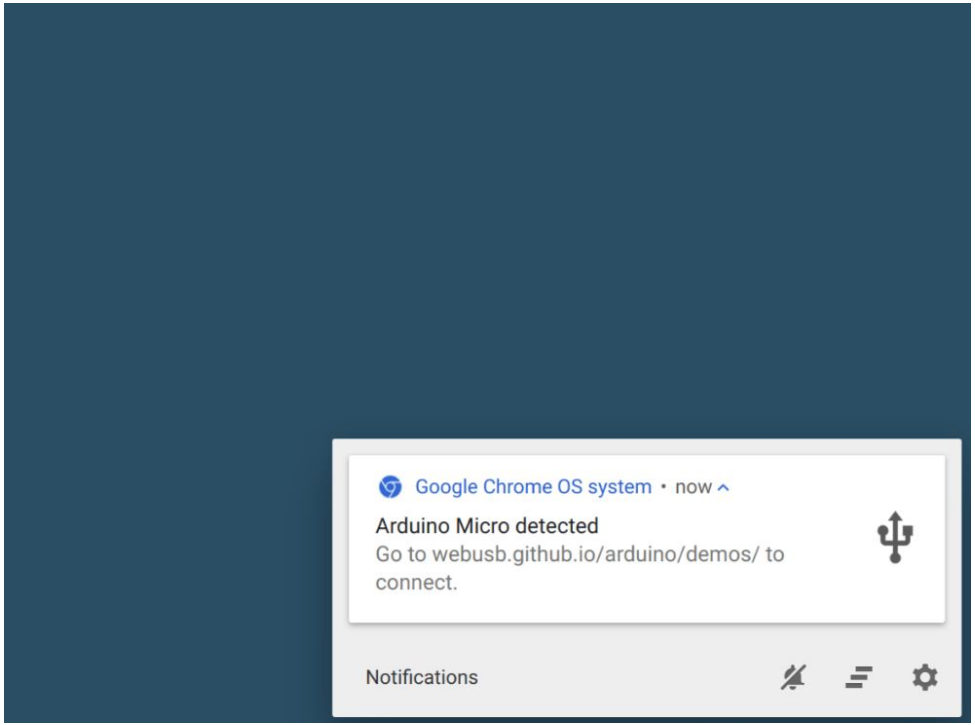
- A board with 802.15.4 radio and USB might be used as a USB adapter
  - Board, radio and USB shall be supported by Zephyr
  - Sample boards: nRF52840 based (reel_board), NXP Freedom-K64F board with NXP Freedom-CR20A shield
- wpanusb Zephyr sample: source, documentation
  - Application define Custom interface and custom USB protocol
  - wpanusb protocol is similar to atusb protocol



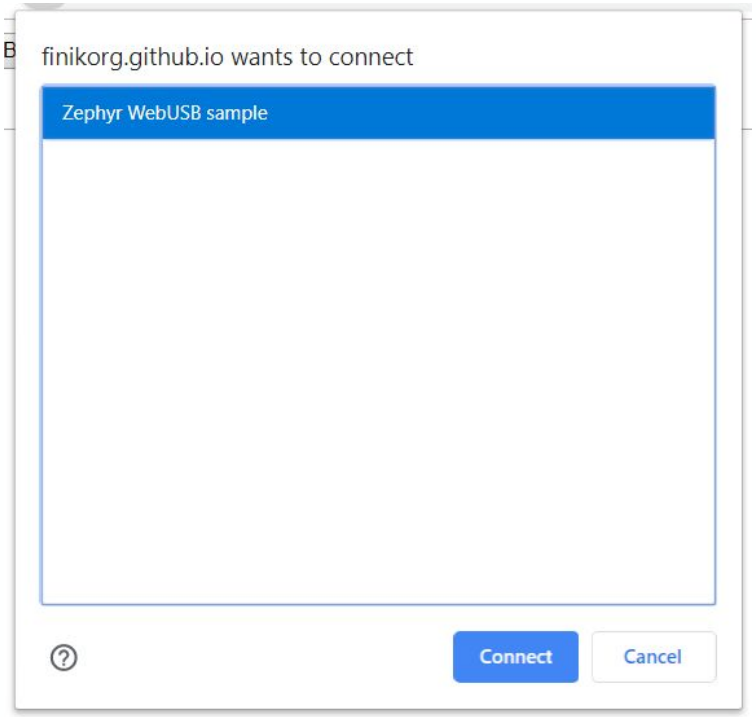
IEEE802.15.4 USB: Linux driver

- SoftMAC driver similar to atusb Linux kernel driver
- Works with wpanusb protocol
- Driver loads on specific Product ID / Vendor ID pair
- Zephyr has own Vendor ID (0x2FE3). Zephyr USB Vendor and Product identifiers.
- Source code: https://github.com/finikorg/wpanusb

WebUSB: Use case



WebUSB Notification when device is connected



WebUSB Device Chooser

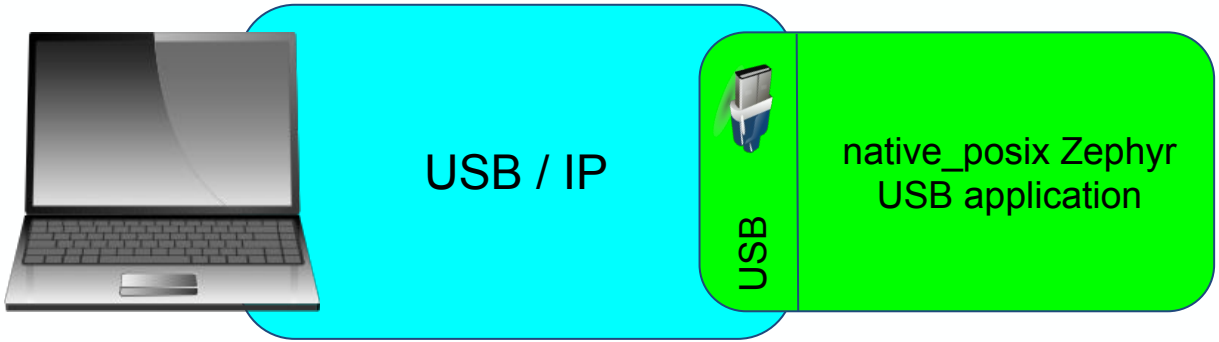
WebUSB: API & Zephyr support

- Simple WebUSB API https://wicg.github.io/webusb/
- Works with recent Chrome and Opera: compatibility table
- USB Device direct access
- USB Device announce support by including special Descriptor
- Vendor - Specific Request specified
- Zephyr supports WebUSB
- Zephyr sample application: source, documentation
    - echoing data from one endpoint to another

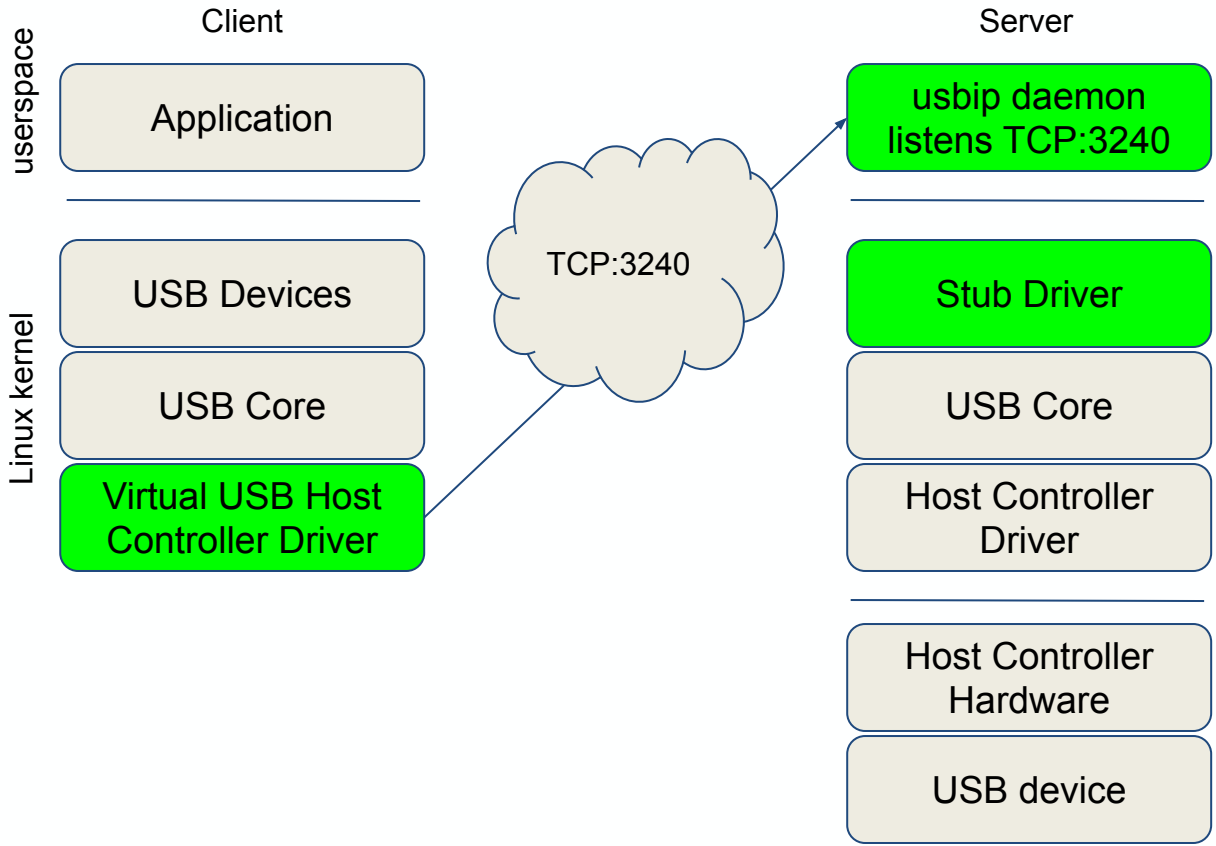
```
// Select configuration #1 for the device
device.selectConfiguration(1)

// Request exclusive control over interface #2
device.claimInterface(2)

// Waiting for 64 bytes of data from endpoint #3
device.transferIn(3, 64)

// Send Control Transfer
device.controlTransferOut({
    requestType: 'class',
    recipient: 'interface',
    request: 0x22,
    value: 0x01,
    index: 0x02})
```

Using virtual USB controller over USB/IP

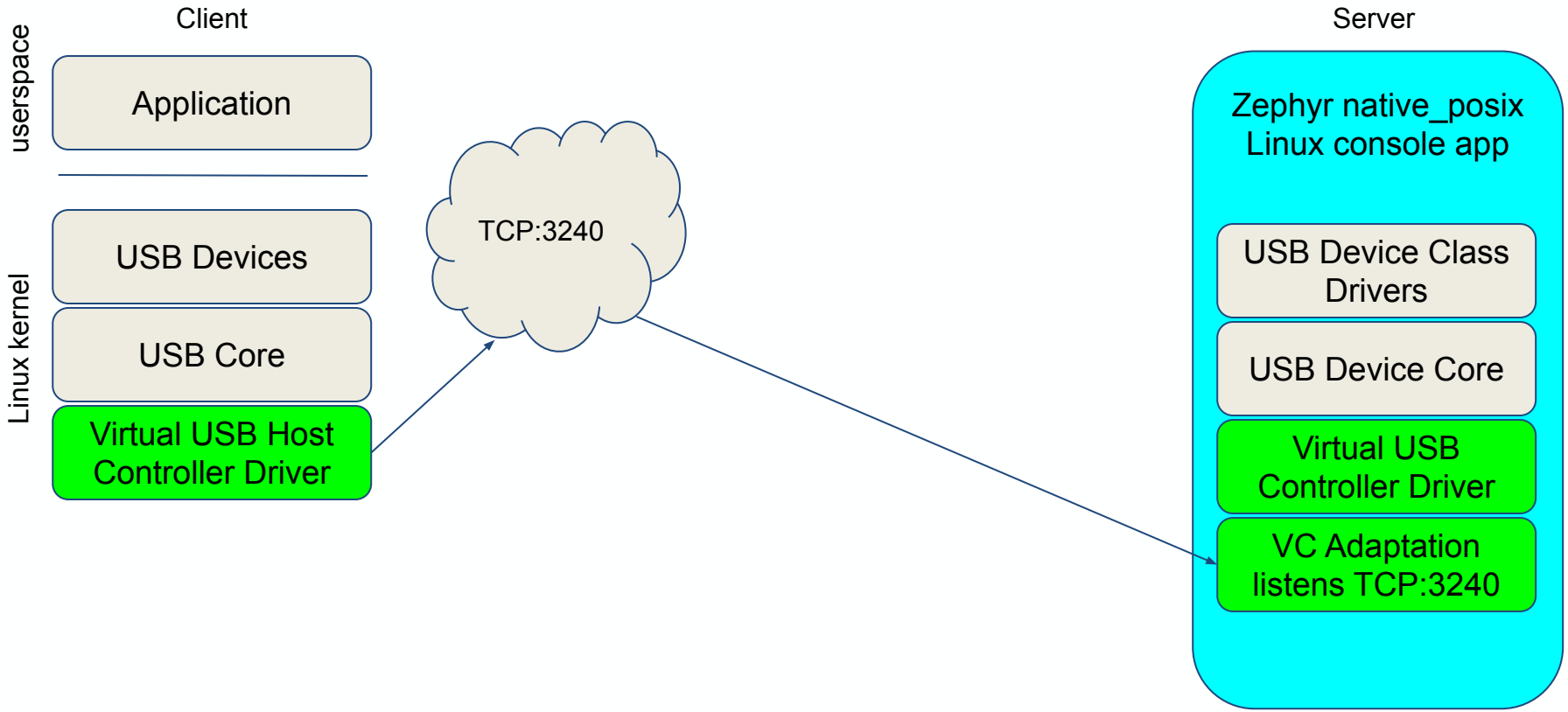
- Method for testing Zephyr USB app in Linux Host
- Zephyr application can be build and run as a native Linux console application
  - native_posix board: https://docs.zephyrproject.org/latest/boards/posix/
- Virtual USB controller over USB/IP protocol
  - Zephyr app works with virtual USB controller connected to the Host
  - Host sees Zephyr device gets connected



Virtual USB Controller over USB/IP



Virtual USB Controller over USB/IP



Zephyr Virtual USB Controller: Attaching

```
$ sudo modprobe vhci-hcd
```

Load USB/IP Virtual Host Controller driver

```
$ west build -b native_posix -t run zephyr/samples/subsys/usb/hid
$ lsof -i -P | grep zephyr
zephyr.ex 13633 niko 4u IPv4 32089735  0t0  TCP *:3240 (LISTEN)
```

Build and run Zephyr app and verify it listens on the USB/IP port

```
$ usbip list -r localhost
Exportable USB devices
======================
 - 127.0.0.1
   1-1: unknown vendor : unknown product (2fe3:0006)
     : /sys/devices/pci0000:00/0000:00:01.2/usb1/1-1
     : (Defined at Interface level) (00/00/00)
     : 0 - Human Interface Device / No Subclass / None(03/00/00)
```

List exportable USB devices with usbip list

```
$ sudo usbip attach -r localhost -b 1-1
```

Attach exportable device to the Host

```
$ lsusb -d 2fe3:0006
Bus 007 Device 013: ID 2fe3:0006
```

Verify USB device attached to the Host

Zephyr Virtual USB Controller: Linux logs

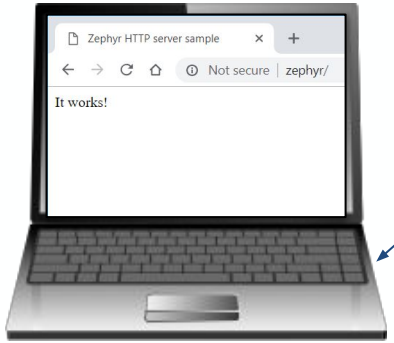
```
vhci_hcd vhci_hcd.0: USB/IP Virtual Host Controller
vhci_hcd vhci_hcd.0: new USB bus registered, assigned bus number 7
vhci_hcd: created sysfs vhci_hcd.0
usb usb7: New USB device found, idVendor=1d6b, idProduct=0002
usb usb7: New USB device strings: Mfr=3, Product=2, SerialNumber=1
usb usb7: Product: USB/IP Virtual Host Controller
usb usb7: Manufacturer: Linux 4.15.0-31-generic vhci_hcd
usb usb7: SerialNumber: vhci_hcd.0
hub 7-0:1.0: USB hub found
hub 7-0:1.0: 8 ports detected
```

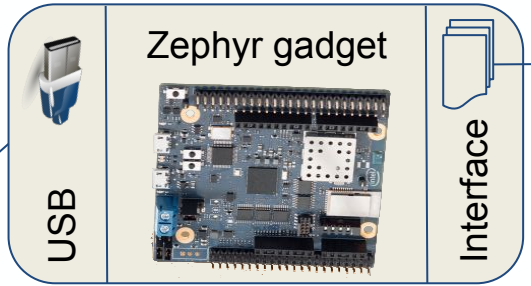
Loading Virtual Host Controller driver

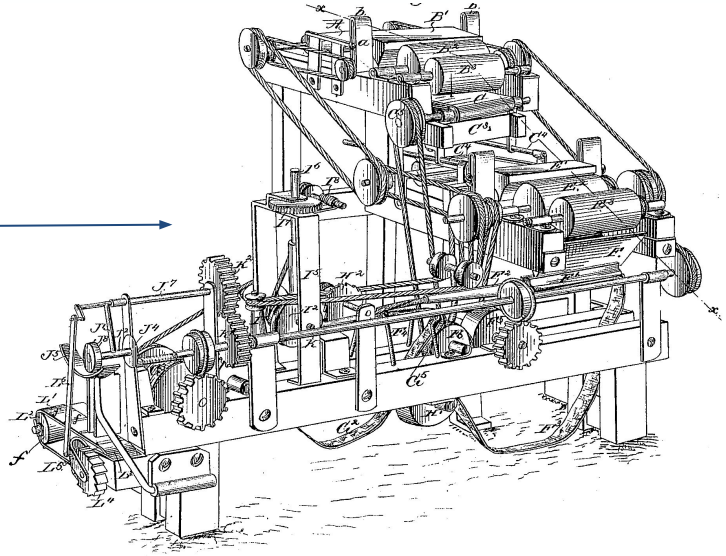
```
usb 7-1: New USB device found, idVendor=2fe3, idProduct=0006
usb 7-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 7-1: Product: Zephyr HID sample
usb 7-1: Manufacturer: ZEPHYR
usb 7-1: SerialNumber: 0.01
input: ZEPHYR Zephyr HID sample as
/devices/platform/vhci_hcd.0/usb7/7-1/7-1:1.0/0003:2FE3:0100.004E/input/input89
hid-generic 0003:2FE3:0100.004E: input,hidraw2: USB HID v1.10 Device [ZEPHYR
Zephyr HID sample] on usb-vhci_hcd.0-1/input0
```

Attaching device over USB/IP

Summary







Complex Hardware

References

Zephyr

- Zephyr OS https://www.zephyrproject.org/
- Zephyr USB Device Stack: documentation
- Native Posix board documentation
- Zephyr dumb HTTP Server app: README, source
- Zephyr wpanusb app: README, source
- Zephyr webusb app: README, source

Contiki

- Contiki OS http://www.contiki-os.org/
- Contiki native border router https://github.com/contiki-os/contiki/tree/master/examples/ipv6/native-border-router

USB

- USB Specifications: https://www.usb.org/documents
- How to Create and Program USB Devices
- Microsoft OS Descriptors: Specifications
- WebUSB API https://wicg.github.io/webusb/
- WebUSB Privacy and Security: Notes

Linux

- USB/IP project http://usbip.sourceforge.net/
- Linux IEEE 802.15.4 Developer's Guide

INTEL OPEN SOURCE TECHNOLOGY CENTER | 01.org